Phase 1: Comparative study of GMRES and BICGSTAB iteration schemes used to solve a discretised two-dimensional Poisson's Partial differential equation.

Phase 2: Study of solving a diffusion equation with different initial conditions

Introduction

Elliptic partial differential equations (PDE) arise naturally from equilibrium or steady-state problem and their solutions, in relation to the calculus of variations, frequently maximize or minimize an integral representing the energy of the system[1]. The best known elliptic equations are Poisson's equation. Poisson's equations are used to model the slow motion of incompressible viscous fluid and the square law theories of electricity, magnetism and gravitating matters[1]. A general Poisson's PDE in two dimensions is given in equation (1).

Equation 1

$$\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}=\frac{\partial u(x,y)}{\partial x}+\frac{\partial u(x,y)}{\partial y}=\nabla^2 u(x,y)=f(x,y)$$

Discrete Poisson's equations are widely used to solve this PDE numerically. A discrete Poisson's equation is the finite-difference analogue of the Poisson's equation. In this scheme, the discrete Laplace operator takes the place of the Laplace operator. And a linear system is solved using iterative methods[1].

Krylov subspace methods are powerful widely used methods for solving large-scale sparse linear systems arising from numerical solutions of PDEs[2]. Given matrix $A \in R^{n\times n}$, and a vector $v \in R^n$, the k-th Krylov subspace generated by them, is denoted $K_k(A,v)$ and is given by $K_k(A,v) = \text{span}\{v, Av, A^2v, \dots, A^{k-1}v\}$. To solve the equation, we let x_0 be the initial guess of the solution. Let $r_0 = b - Ax_0$ be the initial residual vector. Krylov subspace method incrementally finds approximate solutions within $K_k(A,v)$. To construct the basis of the $K(A,v)$ two common methods are used: Arnoldi iteration and the bi-Lanczos iteration. The Arnoldi iteration construct s orthogonal basis of the subspace $K_k(A,v)$. It first starts from a unit vector $q_1 = \frac{v}{\|v\|}$, and iteratively constructs $Q_{k+1} = [q_1|q_2|\dots|q_k|q_{k+1}]$ with orthonormal columns by solving $h_{k+1}q_{k+1} = Aq_k - h_{1k}q_1 - \cdots - h_{kk}q_k$, where $h_{ij} = q_i^T A q_j$ and $h_{k+1,k} = \|Aq_k - h_{1k}q_1 - \cdots - h_{kk}q_k\|$.

The Bi-Lanczos iteration, uses a three-term recurrence. Unlike Arnoldi, the basis will not be orthogonal and it uses two matrix-vector multiplications per iteration. Two widely used iteration methods are Generalised minimal residual method (GMRES)[3] and Biconjugate gradient stabilized method (BICGSTAB)[4]. They are both Krylov subspace methods. GMRES uses Arnoldi iteration, but BICGSTAB uses the Bi-Lanczos iteration.

In the first phase of this project, we consider the equation (2)

Equation 2

$$-\nabla.(\sigma(x,y)\nabla)u(x,y) = f(x,y)$$

where$(x,y) \in [0,1] \times [0,1]$. And f is a real valued function. The ∇^2 is the Laplacian operator. We impose Dirichlet boundary conditions on u. And $\sigma(x,y) = \sigma$ are randomly distributed values.

We use central difference finite-difference method on a 2-dimensional grid to solve the PDE. We will also use a polynomial function for the σ values. We will then assess the validity of our finite-difference solution with a Finite-element solution of the PDE with the polynomial σ with the same boundary conditions using FEniCS[5]. Lastly, we will use several iteration methods used to solve the PDEs and investigate their convergence and performance.

Parabolic PDEs are type of PDEs that are used to describe a wide range of time-dependent processes, including heat conduction, particle diffusion and a range of financial mathematics used especially for derivative pricing[1]. The diffusion equation is a parabolic PDE which describes density fluctuations in a material undergoing diffusion[1].

For the second phase of this project, we consider a diffusion equation (3) of the form

Equation 3

$$u_t = \nabla.(\sigma(x,y)\nabla)u$$

with the same space domain as the elliptic equation in the first phase and time t starting at 0. Initial conditions are given as $u(x,y,0) = g(x,y)$ at time $t = 0$ and $u(x\,y,t) = 0$ on the boundary for all $t \geq 0$. For time-discretisation we use a simple forward difference[2].

We will experiment with three different initial conditions. The first initial condition is a Gaussian centred in the middle with variance 0.1. The second scenario is when the initial condition is the sin of the Gaussian as in the first scenario. And the third scenario is when the initial condition is such that all values are one. We will investigate the solutions of the equation for each initial condition at several time-steps. We will investigate the rate of the diffusion. And lastly using multiple experiments we try to find a relationship between the spatial-step size and temporal-step size in order to ensure the stability of our method.

Methods

First, we approximate the left hand-side of the equation using central difference scheme:

$$\nabla.(\sigma(x,y)\nabla)u(x,y)$$

$$\approx \frac{(\sigma_{i+\frac{1}{2},j}\frac{(u_{i+1,j}-u_{i,j})}{h}) - (\sigma_{i-\frac{1}{2},j}\frac{(u_{i,j}-u_{i-1,j})}{h})}{h}$$

$$+ \frac{(\sigma_{i,j+\frac{1}{2}}\frac{(u_{i,j+1}-u_{i,j})}{h}) + (\sigma_{i,j-\frac{1}{2}}\frac{(u_{i,j}-u_{i,j-1})}{h})}{h}$$

where $\sigma_{i+\frac{1}{2},j}, \sigma_{i-\frac{1}{2},j}, \sigma_{i,j+\frac{1}{2}}, \sigma_{i,j-\frac{1}{2}}$ are approximated in the following ways:

$$\sigma_{i+\frac{1}{2},j} \approx \frac{1}{2}(\sigma_{i+1,j} + \sigma_{i,j}) \text{ and } \sigma_{i-\frac{1}{2},j} \approx \frac{1}{2}(\sigma_{i-1,j} + \sigma_{i,j})$$
$$\sigma_{i,j+\frac{1}{2}} \approx \frac{1}{2}(\sigma_{i,j+1} + \sigma_{i,j}) \text{ and } \sigma_{i,j-\frac{1}{2}} \approx \frac{1}{2}(\sigma_{i,j-1} + \sigma_{i,j})$$

In the iterative systems, I solve the linear system: $A = ub$. Where here the A is the load matrix, b are the boundary values and the u is the vector of unknowns holding value of each point in the grid I'm solving for. The matrix A is defined as:

$$A = \begin{bmatrix} B & D & 0 \\ E & B & D \\ 0 & E & B \end{bmatrix}$$

where B is a tri-diagonal matrix and D and E are diagonal matrices as defined below:

$$B = \begin{bmatrix} a & b & 0 & \cdots & 0 \\ c & a & b & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & b \\ 0 & \cdots & 0 & c & a \end{bmatrix}$$

where $a = -\left(\sigma_{i+\frac{1}{2},j}, +\sigma_{i-\frac{1}{2},j} + \sigma_{i,j+\frac{1}{2}} + \sigma_{i,j-\frac{1}{2}}\right)$ and $b = \sigma_{i+\frac{1}{2},j}$ and $c = \sigma_{i-\frac{1}{2},j}$

$$D = \begin{bmatrix} \sigma_{i,j+\frac{1}{2}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{i,j+\frac{1}{2}} \end{bmatrix}$$

$$E = \begin{bmatrix} \sigma_{i,j-\frac{1}{2}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{i,j-\frac{1}{2}} \end{bmatrix}$$

For the right-hand side, we use a vector f with the number of rows/columns (considering square grid), and all values are 1. And, $b = -h^2 f$ where $h = \frac{1}{mesh\ size\ (i.e. row\ number)}$. We then implement this scheme in PyOpenCL, in a matrix-free way. In the implementation, we take a vector u of unknowns and the implementation then applies the discretisation scheme. We turned this function into a linear operator to enable us to apply several of the iteration schemes and investigate their convergence and computation cost.

For the second phase, we use a forward difference scheme for time discretisation of our parabolic PDE, as given below:

$$u_t \approx \frac{u(x, y, t + \Delta t) - u(x, y, t)}{\Delta t}$$

For the spatial discretisation, we use the same scheme as phase 1 for the elliptic equation. Writing down the solution in terms of its components we can have:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\Delta t}{(\Delta x)^2}\left(\left(\left(\sigma_{i+\frac{1}{2},j}\left(u_{i+1,j}^{n} - u_{i,j}^{n}\right) - \sigma_{i-\frac{1}{2},j}\left(u_{i,j}^{n} - u_{i-1,j}^{n}\right)\right) + \left(\sigma_{i,j+\frac{1}{2}}\left(u_{i,j+1}^{n} - u_{i,j}^{n}\right) - \sigma_{i,j-\frac{1}{2}}\left(u_{i,j}^{n} - u_{i,j-1}^{n}\right)\right)\right)\right)$$

where, Δt is the step-size, Δx is the spatial step size, n is the step number, and all other terms are similar to the ones defined in phase 1. All the implementation is done same as phase 1.

Results

Phase 1:

We implemented the finite difference method on OpenCL. The (Figure 1) shows our finite-difference method used to solve the PDE with random and polynomial σ ($\sigma(x, y) = 1 + x^2 + y$) on a gird of 100 times by 100. The (Figure 2) shows the finite-element method with triangular elements used to solve the PDE with polynomial sigma on a grid of 100 times by 100, using FEniCS. As we can see the solutions are similar which shows that our implementation is correct. (GMRES was used for both finite-difference and finite-element)

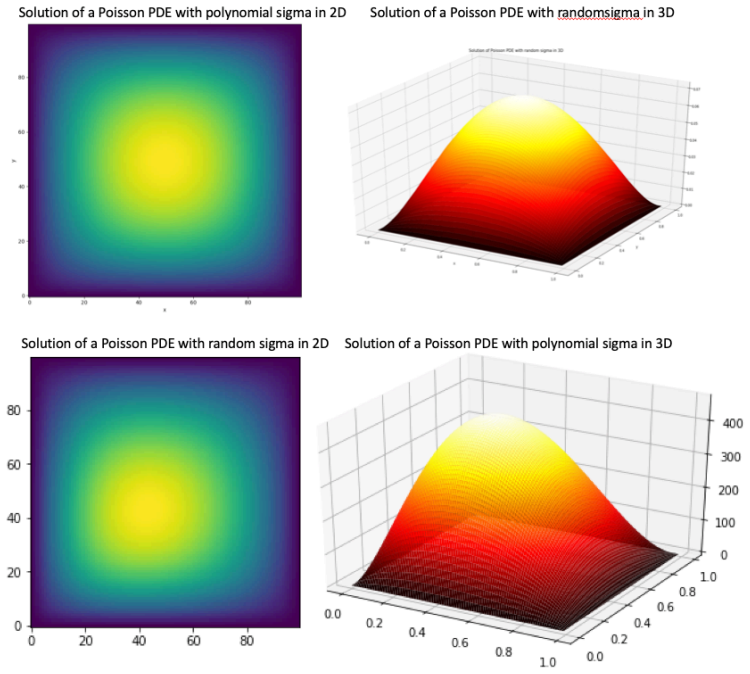

Figure 1. Solution of finite-difference method used to solve Poisson's equation.
In top, σ was randomly distributed. In bottom, σ was distributed with a polynomial function ($\sigma(x, y) = 1 + x^2 + y$).

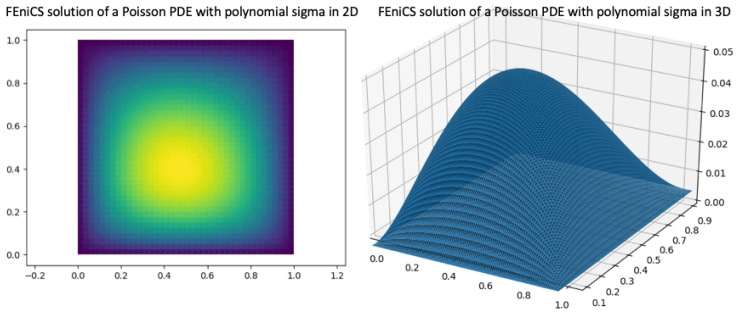

Figure 2. Solution of finite-element method used to solve Poisson's equation.
The value of σ was distributed with a polynomial function ($\sigma(x, y) = 1 + x^2 + y$). This was solved on FEniCS platform

We then used GMRES and BICGSTAB iteration methods to solve the linear system. As the (Figure 3) shows for different mesh sizes (5-70), the log of residuals has been plotted against the iterations. We can see that the GMRES requires less iteration for smaller mesh sizes until about mesh size of 30, where the GMRES starts to require significantly more iterations as the mesh size gets bigger. Furthermore, the plots are logarithmic, the linearity of the plots for the GMRES convergence shows that relation between the mesh size and number of iterations required for convergence is exponential. This relation is linear for the BICGSTB method.

We also measured the timing performance of the methods (Figure 4). Consistent with the convergence plots, the GMRES takes less time for small mesh sizes until the size of 30. After that the BICGSTAB takes significantly many times less time such that for mesh size of 70, it takes about a third of the GMRES method.

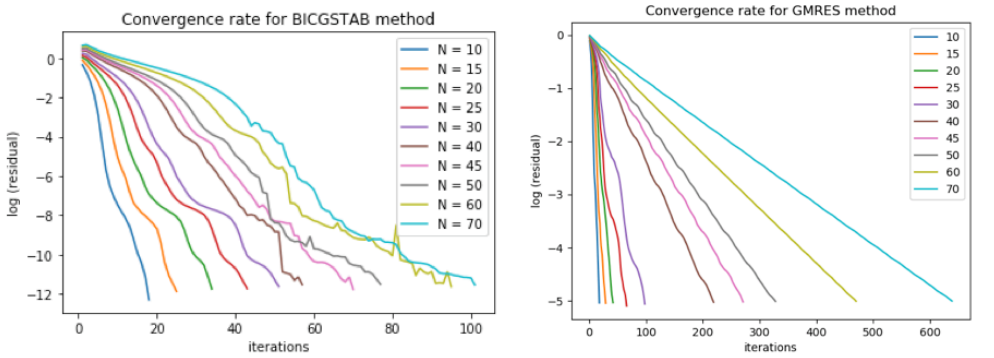

Figure 3. Convergence plots for GMRES and BIOCGSTAB methods.
The logarithm of residuals is plotted against the number of iteration for several mesh sizes (different colours). BIOCGSTAB (left) and GMRES (right).

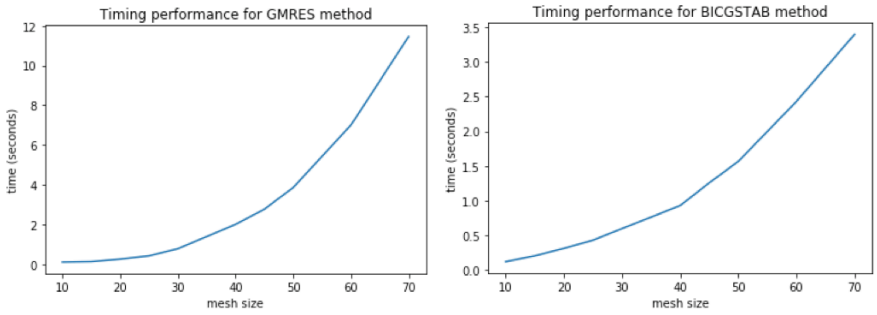

Figure 4. Timing performance of GMRES and BIOCGSTAB methods.
The total time taken for the iteration methods is plotted against the mesh size. BIOCGSTAB (right) and GMRES (left).

Phase 2:

We implemented the finite-difference method for random σ values in OpenCL. We used a grid size of 100 and different time-steps. It should be noted that we used $\Delta t = (\Delta x)^2 \times C = \left(\frac{1}{N}\right)^2 \times C$ for time-step size(Δt) and spatial-step size (Δx). For the value of C we used $\frac{1}{6}$. This value worked and gave stable values throughout. As later is shown this value was suitable for giving stable solutions. For initial conditions, we experimented with three different kinds of initial conditions. In (Figure 5) we used initial condition as a Gaussian distribution centred in the middle with variance of 0.1. In (Figure 6) we used initial condition as the sine of the Gaussian from (Figure 5). In (Figure 7) we used initial condition with values of all 1. As the plots show, the shape of each solution with any of the initial conditions, after a certain time-step starts to look similar to the solution of the Gaussian solution as in (Figure 5).

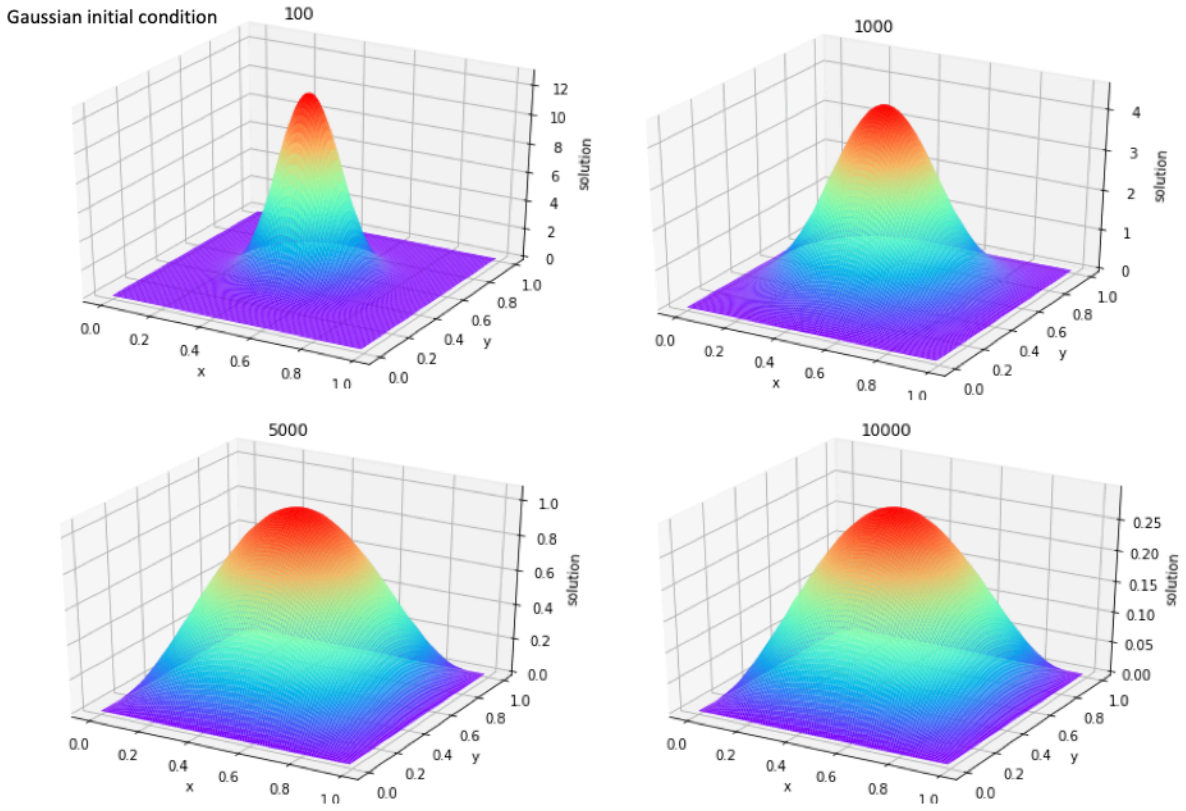

Figure 5. Solution of the diffusion PDE with Gaussian initial condition over time.
The 3D plot of the solutions calculated as the time-step increase. Grid size is 100. Time-steps are written above the plots.

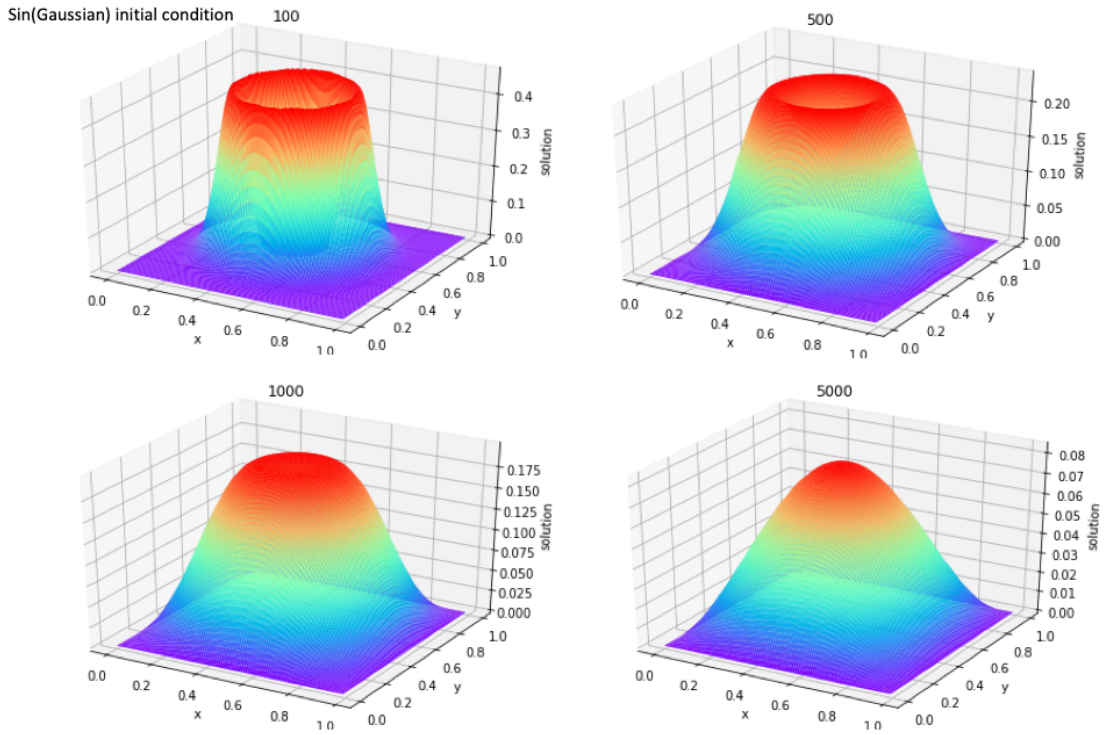

Figure 6. Solution of the diffusion PDE with sin(Gaussian) initial condition over time.
The 3D plot of the solutions calculated as the time-step increase. Grid size is 100. Time-steps are written above the plots.

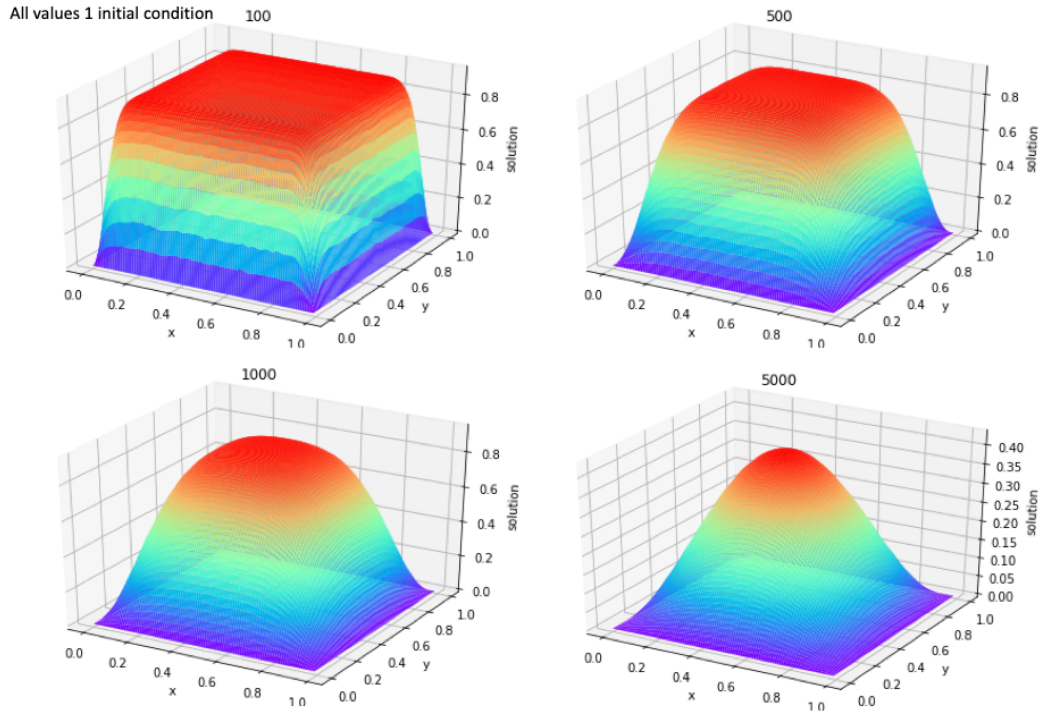

Figure 7. Solution of the diffusion PDE with 1 valued initial condition over time.
The 3D plot of the solutions calculated as the time-step increase. Grid size is 100. Time-step are written above the plots.

To find the rate of the diffusion, we plotted the log of the norm of the solutions vs time steps for each initial condition type (Figure 8). As the plot shows, since the y-axis is logarithmic, the linearity of the plots show that for all the initial conditions, the rate of diffusion is exponential.

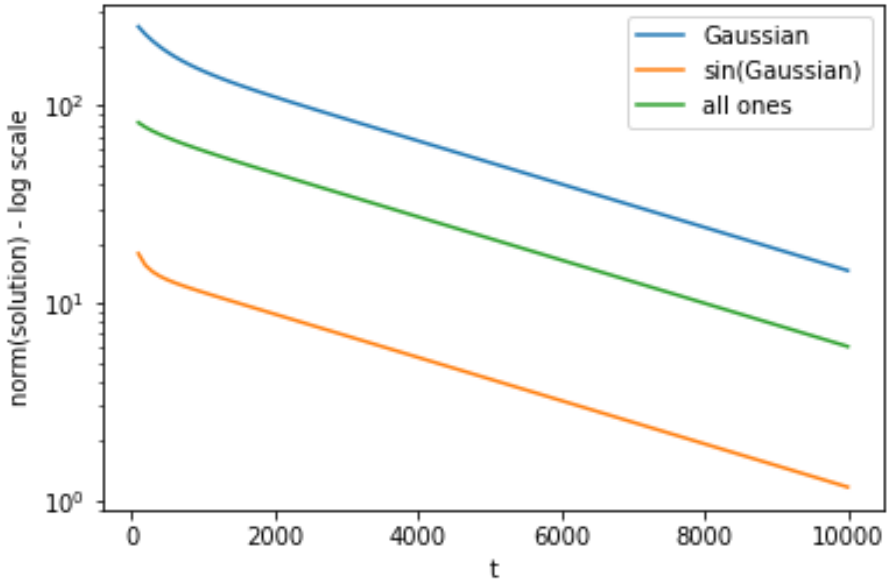

Figure 8. Rate of diffusion with the three initial conditions.
The norm of solutions of the PDE at each time-step (t) was calculated and its log was plotted against the time-steps (t). Grid size is 100.

When we experiment with and increase the time-step size while keeping the grid size fixed, we can see that our finite-difference scheme eventually becomes unstable, like (Figure 9, left). Thus, we are interested in the relation between the time-step size and grid size or the spatial size to maintain the stability of our method.
From the way, our method is laid out, we have a relationship between temporal and spatial step size such that $\Delta t \leq C.(\Delta x)^2$ where C is some positive constant. It should be noted that the $\Delta t = (\Delta x)^2.C = \left(\frac{1}{N}\right)^2.C$
To find an approximation for the numerical value of C, I fixed the grid size and time-steps and used a range of values of C between 0.1 and 1. I repeated the same experiment with different grid sizes and time-steps. By assessing the stability of solutions from the generated plots, an example (Figure 9), we found the value of ~0.25 to be the maximum value of C for the solution to be stable.

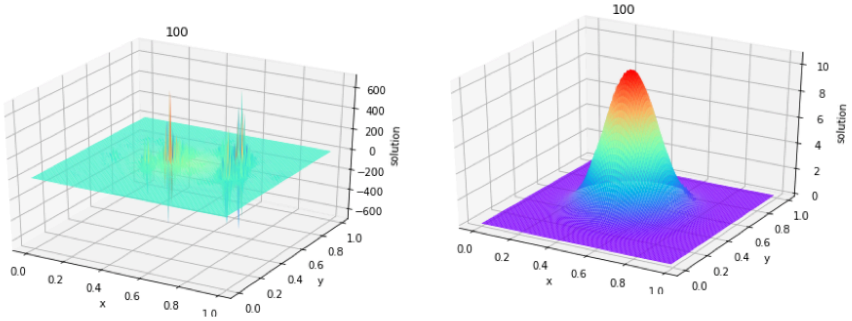

Figure 9. Solution of the diffusion PDE
Solution of the PDE with grid size 100 and time-step 100. The value of C=0.26(left) and C=0.24(right).

Conclusion and Discussion

Phase 1:

In the first phase of this study, we implemented a (finite) central difference method for discretising a 2D Poisson partial differential equation with Dirichlet boundary conditions. We showed the accuracy of our method by comparing it with a Finite element method implementation done on FEniCS project. We then investigated the convergence and speed of GMRES and BICGSTAB, two Krylov subspace iteration methods relying on two different iteration methods, Arnoldi iteration method and bi-Lanczos iteration method[6].

As the convergence plots show, the BICGSTAB converges faster than GMRES for big mesh sizes. This is because the BICGSTAB method enjoys a three-term recurrence, so it outperforms GMRES when many iterations are needed. However, as the convergence plot for the BICGSTAB shows, the convergence starts to become non-smooth as the mesh size starts to get higher. So, the results of this study cannot be used to generalise the performance of BICGSSTAB for very large mesh sizes. In practice this may be solvable if a right-preconditioner is used[6]. Furthermore, there are many widely used existing software packages used for these purposes such as PETSc which supports both left and right pre-conditioning for GMRES and BICGSTAB.

And lastly, we investigated the timing performance of the two methods. The BICGSTAB method outperformed the GMRES method as the mesh size became larger. Obviously, since BICGSTAB requires less iterations than GMRES, it would take less time. One interesting was that for mesh size of 70, GMRES took about 600 iterations and BICGSTAB took about 100 iterations (about a factor of 6 difference). However, for the same mesh size it took GMRES 12 seconds and BICGSTAB took 3.5 seconds (about a factor of 3). So, there is a difference (about a factor of 2) between the computation cost of GMRES and BICGSTAB for each iteration. One explanation for this is that each Arnoldi iteration (used by GMRES) requires the equivalent of one matrix vector multiplication, while bi-Lanczos (used by BICGSTAB) uses the equivalent of two matrix vector multiplication. Hence, the GMRES is twice faster of each iteration. We conclude that GMRES is faster than BICGSTAB, but BICGSTAB out-performs GMRES if many iterations are needed.

Phase 2:

In the second phase of this study, we solved a diffusion equation using the previous method for spatial discretisation and a simple forward difference method for the temporal discretisation. We used experimented with three different initial conditions. All three cases ended up converging to a trajectory that looks like the Gaussian case. The detailed explanation for this is beyond scope of this project, but a short explanation for this is that we can re-write the parabolic equation as $\frac{\partial u}{\partial t} = Au$ where A is the Laplace operator (square matrix of size $N^2 \times N^2$). We can use v and λ as eigenvector and eigenvalues of A so that we can have $u(t) = e^{\lambda t} v$ to be the solution. So, the solution at a certain time-step can be written as a linear combination of exponential functions. The Gaussian distribution is part of

exponential distributions that are generated by exponential functions[8]. So, this explains why the solutions ended up like the trajectory of the one with Gaussian distribution[8].

Furthermore, we found the rate of diffusion to be exponential. This can also be explained by the previous explanation of why the solutions ended up looking like the one with Gaussian initial conditions. Also, as the plot showed, the rate of diffusion goes perfectly exponential after a certain time-step which makes it look identical to the Gaussian case. Given that Gaussian distributions are part of exponential family of distributions this would be expected[8].

And lastly, using a trial and error approach we found a relation between the temporal step size (Δt) and spatial step size (Δx) given a fixed grid size. The Forward-Euler method used in this project is easy to implement, however the drawback is this encountered conditional stability and the requirement that the time step-size is small enough to get stable results. To overcome this, we can use an implicit method like Backward-Euler method[2] which has the drawback of solving a linear system in each step, or perhaps given enough resources we could use a second order method like Crank-Nicolson which is unconditionally stable[2].

References:

1. Smith, G. (1971). Numerical solution of partial differential equations. London: Oxford University Press.
2. Saad, Y. (2003). *Iterative methods for sparse linear systems*, second edition. Philadelphia: Society for Industrial and Applied Mathematics.
3. Saad, Y. and Schultz, M. (1986). *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM Journal on Scientific and Statistical Computing*, 7(3), pp.856-869.
4. Van der Vorst, H. (1992). Bi-CGSTAB*: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems*. SIAM Journal on Scientific and Statistical Computing, 13(2), pp.631-644.
5. M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes and G. N. Wells. (2015) *The FEniCS Project Version 1.5*. Archive of Numerical Software, vol.
6. Ghai, A, Lu,C, Jiao, X. (2018). A comparison of Preconditioned Krylov Subspace methods for Large-scale Nonsymmetric Linear Systems. SIAM Journal on Scientific computing. 15(2).
7. Evans, L. (2010). *Partial differential equations*. Providence, R.I.: American Mathematical Society.
8. Bishop, C. (2010). *Pattern Recognition and Machine Learning*. 1st ed. p.Chapter 2.